頂点順序の最適化によるスケーラブルなグラフ並列処理

頂点順序の最適化によるスケーラブルなグラフ並列処理

新井　淳也† 　塩川　浩昭† 　山室　　健† 　鬼塚　　真††

† 日本電信電話株式会社　〒 180–8585 東京都武蔵野市緑町 3–9–11
†† 大阪大学　〒 565–0871 大阪府吹田市山田丘 2–1

あらまし　Personalized PageRank (PPR) をはじめとするグラフ分析技術には情報検索や推薦など様々な応用があり，分析処理の高速化が求められている．しかしながら，グラフ処理は局所性の低いメモリアクセスパターンを示すためキャッシュ効率が悪く，共有メモリ環境での並列処理時にメモリ帯域が飽和し低いスケーラビリティを示すという問題がある．本論文では階層的なクラスタ構造を捉えたグラフデータのレイアウト変換と圧縮による，グラフ並列処理の高速化手法を提案する．既存のレイアウト手法である RCM と比較し，提案手法は PPR を約 2 倍高速化することが実験により確認された．

キーワード　グラフ処理, 並列処理, 疎行列, キャッシュ, リオーダリング

1. はじめに

Web グラフやソーシャルグラフのように自然発生するつながりを表現したグラフは**実世界のグラフ**と呼ばれ，分析を行うことで学術的，産業的に有用な様々な知識を発見することができる．例えば代表的な分析アルゴリズムである *Personalized PageRank* (*PPR*) [17] を用いると頂点同士の関連度を得ることができる．PPR は個人の嗜好に合わせた検索 [17]，商品の推薦 [20]，画像のタグ付け [27]，語義判別 [5]，文書要約 [21] などに幅広く応用されている．検索や推薦においては迅速なクエリ応答が求められ，他の応用においても処理中に繰り返し PPR を用いる場合があるため，高速な分析処理が必要とされる．ところが，World Wide Web やソーシャルネットワーキングサービスから生成されるグラフは兆単位のエッジ数にまで大規模化しており，このような巨大なグラフに対する分析には長い時間を要してしまう．そこで，近年普及しているマルチコア CPU 上での並列処理によって高速化を図る研究が行われてきた [15, 19, 22].

並列グラフ処理のスケーラビリティはメモリ帯域の飽和によって劣化することが知られている [23]．グラフ処理は一般に局所性の低いメモリアクセスを行うためキャッシュミス率が高く，メモリへのアクセスが頻繁に発生する．局所性の低さは PPR においても確認することができる．PPR は数学的には $n \times n$ 隣接行列 A の固有ベクトル $\boldsymbol{x}$ を求める問題として定義され，$\boldsymbol{x}_{k+1} = A\boldsymbol{x}_k$ を $\boldsymbol{x}$ が収束するまで繰り返す反復法が解法として一般的に用いられる．実世界のグラフの隣接行列は疎行列であるため，$A\boldsymbol{x}_k$ は**疎行列ベクトル積** (sparse matrix-vector multiplication, *SpMV*) となり，次式のように計算される:

$$\boldsymbol{x}_{k+1}[i] = \sum_{j \in \mathbb{N}(i)} A[i,j]\,\boldsymbol{x}_k[j] \ \text{ for } \ i \in [0, n) \qquad (1)$$

ただし $\mathbb{N}(i)$ は頂点 ID が i である頂点と隣接する頂点の集合である．$\mathbb{N}(i)$ は疎であるため，式 1 において $\boldsymbol{x}_k[j]$ に対するアクセスは低い局所性を示す．例えば $N(100) = \{9, 833, 8534\}$ であれば $\boldsymbol{x}_k[9]$, $\boldsymbol{x}_k[833]$, $\boldsymbol{x}_k[8534]$ というように，メモリ上で散らばった領域へのアクセスが行われる．このようなアクセスはキャッシュミスを発生させ，メモリ帯域消費の増大によりスケーラビリティの低下を引き起こす．

グラフ処理および SpMV の局所性を高める手法として *Reverse Cuthill-McKee* (*RCM*) [14] などによる**リオーダリング**が広く用いられている [9,12,16,18,28]．グラフにおけるリオーダリングとは，頂点 i に隣接する頂点が i と近い ID 番号を持つように頂点 ID を振り直す操作である．このような頂点 ID の振り直しを行うことで，式 1 における $\boldsymbol{x}_k$ へのアクセスはより高い局所性を示すようになる．例えば $\mathbb{N}(100) = \{99, 101, 102\}$ であればアクセス先は $\boldsymbol{x}_k[99]$, $\boldsymbol{x}_k[101]$, $\boldsymbol{x}_k[102]$ というように，メモリ上で近傍の領域となる．局所性が向上することでメモリ帯域の飽和が緩和され，スケーラビリティの改善につながる．

RCM など伝統的なリオーダリングアルゴリズムは現在も一般的に利用されているが，それらの手法は実世界のグラフが示す性質を捉えていない．近年提案されたアルゴリズムである *Layered Label Propagation* (*LLP*) [6] は，実世界のグラフが示す高いクラスタ性を利用することにより局所性の向上を達成している．LLP によるリオーダリングでは，まずクラスタリングによってグラフをクラスタ，即ち密に相互接続された頂点の集合へ分割する．次に各クラスタに含まれる頂点へ連続した ID 番号を割り当て直す．クラスタ内の頂点は密に接続されているため，クラスタのメンバが連続 ID を持つことは隣接する多くの頂点が自身と近い ID 番号を持つことを意味し，局所性の向上をもたらす．しかしながら，LLP は実世界のグラフが持つ性質を捉えているものの，現代のプロセッサが持つキャッシュ階層 (L1, L2, L3 など) は捉えていない．ワーキングセットを小さく抑えることでコアに近く低レイテンシな階層のキャッシュにデータを配置させる最適化は，密行列演算などにおいて一般的である．このようにキャッシュ階層の性質を捉えることは性能に大きな影響を及ぼす．

そこで本論文では，2 つの階層に着目した新しいリオーダリングアルゴリズムを提案する．2 つの階層とは，(i) プロセッサ

のキャッシュ階層と (ii) 実世界のグラフが内包するクラスタの階層構造である．階層的なクラスタは様々なネットワークにおいて生じ得る．例えば生徒のソーシャルネットワークでは，同じ学校の生徒同士に深い結びつきが見られ，さらに同じ学年，同じクラスというように一層結びつきが強くなっていくと考えられる．同様に大学の Web サイトから作られる Web グラフでも，大学，学部，学科というように階層的に結びつきが強くなるのが自然である．このようなクラスタの階層とキャッシュの階層を対応付けることによって，小容量だが低レイテンシなキャッシュメモリの活用を狙う．具体的には，まず塩川ら [31] によって提案されているような階層的グラフクラスタリングアルゴリズムを用いて階層的なクラスタを抽出し，次に全ての階層においてクラスタのメンバが連続した ID を持つように ID の再割り当てを行う．生徒の階層的クラスタの例でいえば，各クラスの生徒が通し番号の ID を持ち，各学年や学校の単位で見ても生徒の ID が通し番号となっているように ID を割り当てる．クラスタの階層を捉えることによってクラスタの一部を L1, L2 キャッシュ上に格納できるようになり，レイテンシの影響が一層低減される．さらに，いずれのキャッシュ階層にも収まらない大規模なクラスタに対してもその一部をキャッシュに格納することが可能となり，メモリアクセスの低減とスケーラビリティの改善がもたらされる．

手法の効果を評価するため，提案するリオーダリングアルゴリズムをグラフデータ圧縮などの最適化と共に実装し，共有メモリのマルチコア環境において実験を行った．リオーダリングの効果を測定するためのグラフ分析手法としては PPR を用いている．提案手法によってリオーダリングされたグラフに対する PPR 処理は高い局所性を示し，RCM および LLP との比較で最大約 2 倍高速である．またリオーダリングに要する時間も LLP と比較し最大 27 倍短縮されている．例えば 1 億頂点規模のグラフでは約 12 分でリオーダリングを完了できることが確認された．

本論文の構成は以下の通りである．2 章で関連研究について述べ，3 章で本論文にて必要となる前提知識を説明する．4 章では提案するリオーダリングアルゴリズムについて述べ，5 章で実装上の最適化について説明する．さらに 6 章で提案手法の効果を評価し，最後に 7 章で本論文のまとめと今後の課題について述べる．

2. 関 連 研 究

PPR は与えられたクエリに対して応答する分析手法であるため，前処理を追加することで後続のクエリ処理を高速化する研究が行われてきた [13, 24, 30]．例えば Fujiwara ら [13] は前処理中に隣接行列を LU 分解することによって高速なクエリ処理を達成した．

しかしながら，これらの PPR に関する前処理手法は部分的に直接解法を使用しており，適用先は PPR またはその他の PageRank から派生した分析手法に限られる．PPR に限らずグラフ処理一般に効果を示す前処理手法として，局所性を向上させるためのリオーダリングが知られている．リオーダリングは元々fill-in を減少させることで線形ソルバの性能を向上させるために用いられてきた [14] が，Gropp ら [16] によってリオーダリングの局所性向上効果が確認されて以降，キャッシュ効率の改善による SpMV の高速化のためにも使用されるようになった．グラフ処理においては Frasca ら [12] がグラフ分析アルゴリズムの一つである Betweenness Centrality [7] を高速化するためにリオーダリングを使用し，並列処理性能を約 2 倍向上させた．

リオーダリングアルゴリズムに関しては 1960 年代から研究が行われている [10] が，近年も新しいアルゴリズムが提案されている．最も有名なアルゴリズムの一つである *Reverse Cuthill-McKee* [14] は次数の低い頂点から順に訪問する幅優先探索を行い，訪問順の逆順に頂点 ID を与える．RCM にはアルゴリズムが単純であるためリオーダリング時間が短く済むという利点がある．しかし RCM は実世界のグラフが示す性質を捉えることができない．Boldi と Vigna によって提案された *Layered Label Propagation* (*LLP*) [6] は実世界のグラフが持つクラスタ性の高さに着目し，クラスタリングを用いたリオーダリングを行う．LLP ではまずクラスタの粒度に関わるパラメタをランダムに変化させながら同じグラフに対して繰り返しクラスタリングを行い，各パラメタで得られた結果を層として記録する．次に，全ての層において同じクラスタに属する頂点は近傍に存在することから，それらの頂点に連続した頂点 ID を振り直す． LLP はグラフ処理の高速化ではなく局所性を利用したグラフの圧縮を目的とした手法だが，結果的に隣接する頂点同士に近い ID が与えられるためグラフ処理の高速化も期待できる．

本研究では高いクラスタ性を想定できる実世界のグラフを対象とし，LLP と同様にクラスタリングの結果を用いたリオーダリングを行う．LLP と異なるのは，提案手法がキャッシュ階層とクラスタの階層を捉え，レイテンシの小さいキャッシュメモリの活用を狙う点である．

3. 事 前 準 備

本章では本論文の高速化対象である PPR，疎行列向けデータ構造である *Compressed Sparse Row* (*CSR*) [29]，およびリオーダリング時に使用する階層的クラスタリング技術について説明する．

3.1 Personalized PageRank

PPR は PageRank [26] と同様，ランダムサーファーモデルに基づく確率計算によって頂点間の関連の深さを数値化する分析手法である．計算時の入力は有向グラフ $G = (\mathbb{V}, \mathbb{E})$ ($\mathbb{V}$ は頂点の集合，$\mathbb{E}$ はエッジの集合)，テレポート係数 c，及びクエリとしての**嗜好ベクトル** $\boldsymbol{q}$ ($|\boldsymbol{q}| = 1$) であり，それらを基に *Personalized PageRank* **ベクトル** (*PPV*) $\boldsymbol{s}$ が出力される．$\boldsymbol{s}$ は式 2 を収束するまで繰り返し計算することで求められる．

$$\boldsymbol{s}_{k+1} = (1 - c)W\boldsymbol{s}_k + c\boldsymbol{q} \tag{2}$$

ここで W はエッジ $(i, j) \in \mathbb{E}$ について $W[i, j] = 1/d^+(j)$ で定義される行列である．ただし $d^+(j)$ は頂点 j の出次数とする．

$$A = \begin{pmatrix} 0 & 0.1 & 0.2 \\ 1.0 & 0 & 0 \\ 2.0 & 2.1 & 0 \end{pmatrix}$$

```
AA = {0.1, 0.2, 1.0, 2.0, 2.1}
JA = {  1,   2,   0,   0,   1}
IA = {  0,   2,   3,   5}
```

図 1　CSR の例

実世界のグラフでは $|\mathbb{E}| \ll |\mathbb{V}|^2$ であるため W は疎行列となる．また c の値は一般的に 0.15 が用いられる [17] ことから，本論文でもそれに倣う．PPR では式 2 の SpMV である $W\boldsymbol{s}_k$ がボトルネックとなるため，本論文ではリオーダリングによる局所性の向上によって高速化を目指す．

3.2　Compressed Sparse Row

疎行列をメモリ上で効率よく表現するために CSR が広く用いられている．CSR は 3 つの配列 AA, JA, IA を用いて疎行列を表現する．例として図 1 に行列 A を CSR で表した．配列 AA には行列 A 中の非零要素が行メジャーで格納されている．配列 JA の k 番目の要素 JA[k] は AA[k] の値が何列目の値であるかを保持する (注1)．配列 IA は AA，JA の要素と行の対応を保持する．i 行目の非零要素は AA[IA[i] . . . IA[$i+1$] − 1] に，非零要素の列番号は JA[IA[i] . . . IA[$i+1$] − 1] に格納されている．

グラフの隣接行列において行数は頂点数 $|\mathbb{V}|$ と，非零要素数はエッジ数 $|\mathbb{E}|$ とそれぞれ等しいため，AA，JA の長さは $|\mathbb{E}|$，IA の長さは $|\mathbb{V}|+1$ となる．また，AA はエッジの重み，JA は隣接頂点の ID に相当する．

3.3　階層的グラフクラスタリング

現在最も高速な階層的クラスタリング手法の一つとして塩川らによる手法 [31] (以降，**塩川法**) が知られている．塩川法は貪欲に**モジュラリティ** [25] の最適化を行うアルゴリズムである．モジュラリティとはクラスタリング結果の良さを表す指標の一つで，エッジの存在がクラスタ内において密，クラスタ間において疎である程高い値を示す．

塩川法は 2 つの工夫によって分析処理を高速化している．

頂点の逐次集約　塩川法はモジュラリティを最も上昇させる隣接頂点へ各頂点を貪欲に集約することによってグラフの規模を徐々に縮小させる．集約された頂点は 1 つのクラスタを代表する頂点となる．

次数順の頂点選択　逐次集約は次数の小さい頂点から順に行われる．逐次集約によってグラフの規模が小さくなっていくため，元々高次数だった頂点も集約の進行に従い次数を低下させていく．従って次数の小さい頂点から順に集約することで隣接頂点の参照回数が減少し高速化する．

具体的な塩川法のステップは次の通りである．ただし，入力は無向グラフ $G = (\mathbb{V}, \mathbb{E})$ とする．

（1）頂点を次数の昇順にソートし，キュー $\mathbb{P}$ に格納する．
（2）$\mathbb{P}$ から頂点を一つ取り出し u と置く．
（3）u の隣接頂点のうち，集約した際のモジュラリティ上昇量 $\Delta Q(u,v)$ が最も大きい隣接頂点を v と置く．$\Delta Q(u,v)$ は次のように定義される:

$$\Delta Q(u,v) = 2\left(\frac{w_{uv}}{2m} - \frac{d_u d_v}{(2m)^2}\right) \qquad (3)$$

ただし，w_{uv} はエッジ (u,v) の重み，m は全エッジの重みの和，d_u, d_v はそれぞれ u と v の重み付き次数である．
（4）もし $\Delta Q(u,v)$ が正なら，頂点 u を v へ集約する．集約後の頂点 v は集約前に u と v が持っていた両方のエッジを併せ持ち，u と v から成る一つのクラスタを代表する頂点となる．
（5）もし $\mathbb{P}$ が空でなければステップ (2) へ戻る．

以上のステップを終了したときグラフ G 上に残っていた頂点がグラフ中のクラスタを代表する集約頂点である．それらの頂点へ集約された頂点を展開することによってクラスタのメンバを得ることができる．

4.　リオーダリングアルゴリズム

本章では我々が提案するリオーダリングアルゴリズムの詳細について説明する．

4.1　基本アイデア

実世界のグラフはクラスタ性が高く，互いに密に接続し合った頂点の集合，すなわちクラスタを抽出できる．さらにそれらのクラスタの中に存在するエッジの粗密を捉えることで，より小さくより密に接続された頂点の集合を階層的に抽出することが可能である．一方現代の CPU もまたキャッシュメモリに階層構造を持ち，コアに近いほどサイズとレイテンシは小さくなる．そこで階層的なクラスタを捉えたリオーダリングによってクラスタの階層とキャッシュの階層の対応を取ることができ，より高速な SpMV およびグラフ処理が可能になると考えられる．

階層構造を捉えたリオーダリングを行うため，階層的クラスタリング手法によって次のようなクラスタ構造を抽出する．まず，$\mathbb{C}_1^{(1)}, \mathbb{C}_2^{(1)}, \ldots \subset \mathbb{V}$ を最も浅い階層にあるクラスタとする．ある階層 n のクラスタ $\mathbb{C}_i^{(n)}$ は階層 $n+1$ の互いに交わらないクラスタの和集合として定義され，$\mathbb{C}_i^{(n)} = \mathbb{C}_{j1}^{(n+1)} \cup \mathbb{C}_{j2}^{(n+1)} \cup \ldots$ と表すことができる．そして，最も深い階層のクラスタは 1 つの頂点だけを含み，全部で $|\mathbb{V}|$ 個ある．

このような階層的クラスタに対して，全階層の全クラスタについてメンバの頂点 ID が連続値となるように新しい ID を割り当てる．つまり $\mathbb{C}_i^{(n)}$ に含まれる頂点の ID が連続値であり，$\mathbb{C}_{j1}^{(n+1)}, \mathbb{C}_{j2}^{(n+1)}, \ldots$ に含まれる頂点の ID も同様に連続値となるようにする．以上のようなリオーダリングによってより深い階層で密に接続された頂点から順に近い ID 番号が割り当てられ，データの参照時によりコアに近いキャッシュ階層でヒットすることが期待できる．

4.2　塩川法を用いたリオーダリング

次に，高速な階層的クラスタリング手法である塩川法を用いて前述の基本アイデアを実施する方法について説明する．

3.3 節のステップ (4) にあるように，塩川法では常に 2 つの頂点を 1 つの頂点へ集約していく．つまり階層 n のクラスタは階層 $n+1$ にある 2 つのクラスタの和集合 $\mathbb{C}_i^{(n)} = \mathbb{C}_{j1}^{(n+1)} \cup \mathbb{C}_{j2}^{(n+1)}$ となる．このような塩川法によるクラスタリングの過程は二分木構造のデンドログラムとして表すことができる．図 2 に例を

(注1)：ここでは C 言語等を想定し 0 列目から開始している (zero-based)．

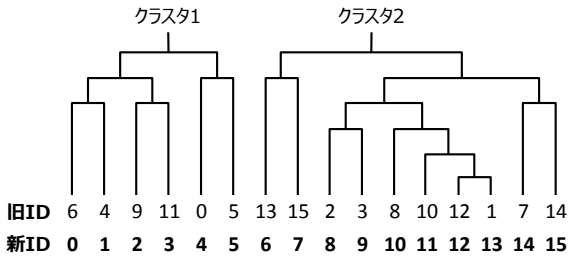


図 2　塩川法によって生成されるデンドログラムの例

示した．このデンドログラムは 16 個の頂点を含むグラフから 2 つのクラスタが抽出される過程を表している．デンドログラムの木構造において，集約された 2 頂点のどちらを左側の子にするか，右側の子にするかは任意であるが，例えば集約されて残る側の頂点を右側の子にするというような規則を設けることはできる．

塩川法で得られた階層的クラスタに対して 4.1 節で述べた ID 割り当てを行うためには，デンドログラムの木構造において深さ優先探索を行い，葉に対して訪問順に ID を与えればよい．ここでいう葉とはデンドログラムの末端にある，集約前の頂点を意味する．図 2 のデンドログラムに対してこのような ID 割り当てを行うと，図中に「新 ID」として表される頂点 ID を得ることができる．いずれの階層のクラスタも連続した新 ID の頂点を含んでおり，階層を捉えたリオーダリングが行われている．

5. 実　　装

本章では実装上の最適化手法について説明する．

5.1 グラフデータ圧縮

近年のグラフ規模の増大に鑑み，実装では頂点 ID を 64 ビット整数で表現する．また，グラフのトポロジは隣接行列として CSR 形式で保持する．ただし並列処理時のメモリ帯域圧迫を緩和するため，リオーダリングの結果を用いて隣接頂点 ID を保持する CSR の配列 JA を圧縮する．リオーダリングを行うと頂点 u とその隣接頂点 v の ID が接近することから，u と v の新しい ID の差は 64 ビットより小さい整数に格納できる可能性が高まる．この性質を利用するため，64 より小さいビット数 β を定め，隣接行列を**遠距離** *CSR* と**近距離** *CSR* の 2 つに分割する．遠距離 CSR は通常の CSR と同じ構造を持ち，ID の差が β ビットより大きい整数となるエッジの情報を保持する．一方，近距離 CSR は ID の差を β ビットで表現可能なエッジの情報を保持するため，配列 JA に β ビットの整数を用いて隣接頂点間の ID の差を格納する．β は 64 より小さいため，これにより JA のサイズを削減できる．最も高い圧縮率を実現する β はグラフデータとリオーダリングアルゴリズムによって変化するが，評価に使用した実装ではヒューリスティックに $\beta = 16$ と定めている．

さらに，PPR の定義を示した式 2 の W はその定義から同じ列に存在する非零要素の値が重複しているため，式 2 を次式に変換し重複した値を排除する:

表 1　評価に用いたデータセット

グラフ名	頂点数	エッジ数	説明
berkstan [3]	685,230	7,600,595	Web グラフ
uk-2002 [2]	18,520,486	298,113,762	Web グラフ
uk-2005 [2]	39,459,925	936,364,282	Web グラフ
webbase [2]	118,142,155	1,019,903,190	Web グラフ
ljournal [3]	4,847,571	68,993,773	ソーシャルネットワーク
orkut [3]	3,072,441	117,185,083	ソーシャルネットワーク
patents [3]	3,774,768	16,518,948	特許の引用ネットワーク

$$\boldsymbol{s}_{k+1}[i] = (1-c)\sum_{j\in\mathbb{N}(i)}\frac{\boldsymbol{s}_k[j]}{d^+(j)} + c\boldsymbol{q}[i] \text{ for } i \in [0, |\mathbb{V}|) \quad (4)$$

これに伴い CSR 表現に求められるのは隣接頂点 ID の保持のみとなるため，AA を除去し JA と IA のみ用いて隣接行列を表現する．このようなグラフデータサイズの削減によってメモリ帯域の飽和を緩和し，スケーラビリティの向上を狙う．

5.2 並 列 処 理

式 4 で表される SpMV の各行の計算を，OpenMP を用いて並列化する．その際負荷分散は 2 つの方針 (i) 各スレッドは連続した行の範囲を担当すること，および (ii) 各スレッドに割り当てられる行の合計非零要素数を均一にすること，に基づいて行う．リオーダリングによって近接する行 i, i' の間で $\mathbb{N}(i) \cap \mathbb{N}(i')$ が増大するため，連続した行の範囲を各スレッドに割り当てることで時間的局所性の高さを活用することができる．さらに SpMV の計算負荷は非零要素数にほぼ比例するため，各スレッドの負荷を均一にするためには割り当てる非零要素数を可能な限り均一にする必要がある．そこで，スレッド $1, 2, \ldots, t$ に割り当てる行の区間を $[\pi_0, \pi_1), [\pi_1, \pi_2), \ldots, [\pi_{t-1}, \pi_t)$ と置き，それぞれの π_i を次のように定める:

$$\begin{cases} \pi_0 = 0 \\ \pi_{i+1} = \arg\min_k \left| \sum_{j=\pi_i}^{k} |\mathbb{N}(j)| - \frac{|\mathbb{E}| - \sum_{j=0}^{\pi_i - 1} |\mathbb{N}(j)|}{t-i} \right| \end{cases}$$

上式はスレッド 1 から $i-1$ までに割り当てられていない非零要素をスレッド i 以降で等分しようとする式である．これにより，理想的な分割である $|\mathbb{E}|/t$ と近い数の非零要素が各スレッドに割り当てられる．

6. 評　　価

提案手法の効果を確認するため，RCM および LLP と性能を比較する．プログラムは C++で実装し，最適化オプション `-Ofast` を指定した GCC 4.9.2 でコンパイルされている．使用したデータセットは表 1 の通りである．データセットの初期頂点順への依存を無くすため頂点 ID をランダムに割り当て直して使用している．また既存研究 [11] と同様，PPR の収束は式 2 において $|\boldsymbol{s}_{k+1} - \boldsymbol{s}_k| < 10^{-8}$ が成立したときとする．6.2 節におけるキャッシュミス回数の計測には Performance Application Programming Interface (PAPI) [8] を，メモリ帯域の計測にはインテル VTune Amplifier XE 2013 [1] を使用した．

実験はソケットを 2 つ持つ non-uniform memory access (NUMA) 環境において行った．NUMA 環境ではメモリが

表 2　実験環境

CPU	Intel Xeon E5-2697v2 12 コア ×2 ソケット
L1 キャッシュ	命令:32KB/コア, データ:32KB/コア
L2 キャッシュ	256KB/コア
L3 キャッシュ	30MB/ソケット
メモリ	256GB PC3-15000
最大メモリ帯域	59.7GB/s
Intel QPI 帯域	片方向 16GB/s，双方向 32GB/s

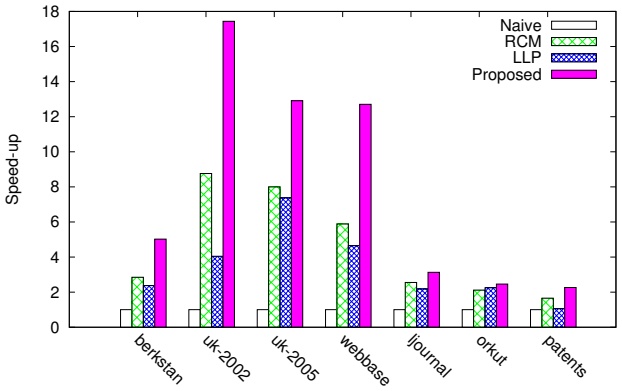


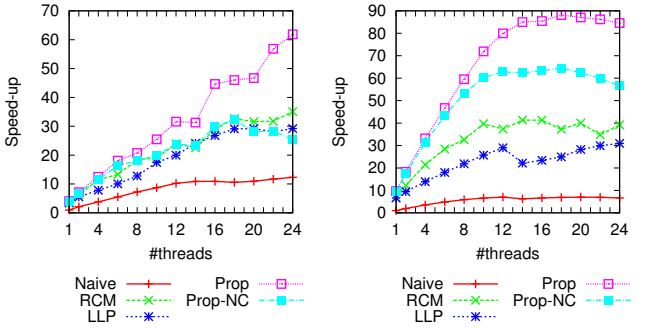
図 3　Naive を 1 とした 24 スレッド時の PPR クエリ処理速度

NUMA ノード (ソケット) に接続されており，各コアは自身のノードに接続されたメモリに対しては高速な直接アクセスが可能である．しかし他のノードに接続されたメモリへアクセスする際はノード間で Intel QPI による通信が必要となり，アクセス速度が低下する．実験ではスレッド 1–12 を NUMA ノード 1 に割り当てるため 12 スレッドまでは NUMA の影響を受けないが，スレッド 13–24 は NUMA ノード 2 に割り当てるため NUMA の影響が生じる．その他の実験環境に関する詳細は表 2 に示した．

6.1　PPR クエリ処理速度

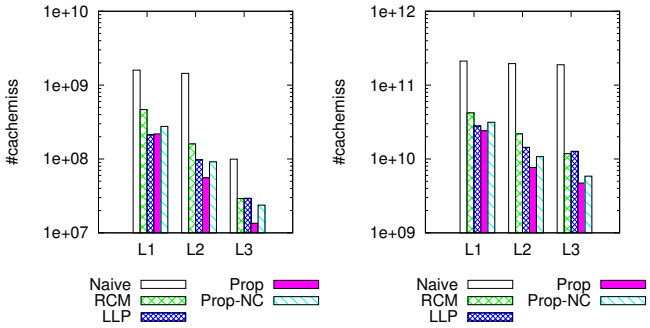
手法の処理速度を評価するため 24 スレッド実行時の PPR クエリ処理時間を計測した (表 3 (a))．“Naive” ではランダムな頂点順序のグラフを使用し圧縮を行わず，“RCM”，“LLP”，“Prop” (Proposed) ではそれぞれ RCM，LLP，提案手法でリオーダリングしたグラフを使用し，5.1 節で述べた圧縮手法を適用している．さらに Naive を 1 として他の手法の処理速度を相対的に表した (図 3)．提案手法はいずれのグラフに対しても最高の処理速度を達成しており，Naive に対しては uk-2002 で最大 17 倍，RCM と LLP に対してはそれぞれ webbase で 2 倍，uk-2002 で 4 倍高速である．一方でグラフ間での処理速度の差も大きい．提案手法はグラフのクラスタ性が高いことを前提としているため，クラスタ性の低いグラフでは性能が低下すると考えられる．実際に，クラスタ係数は高速化率の高い berkstan で 0.60，逆に高速化率の低い patents で 0.08 である [3] ことから，実験結果からもこの傾向を読み取ることができる．

さらに berkstan と webbase についてスレッド数 1–24 における PPR クエリ処理のスケーラビリティを調査し，1 スレッド時の Naive を 1 として相対的な高速化率を図 4 に示した．なお図中の “Prop-NC” (Proposed-No-Compression) では圧縮を行っ



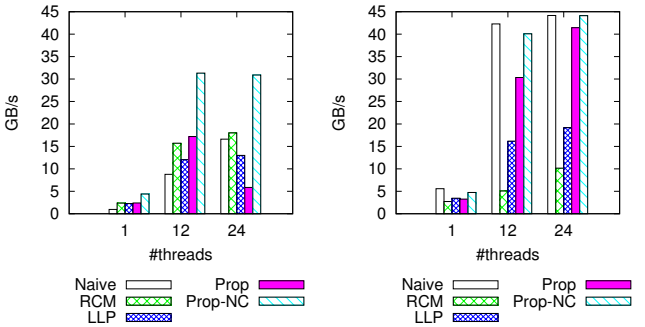
(a) berkstan　(b) webbase

図 4　1 スレッド時の Naive を 1 とした PPR クエリ処理高速化率



(a) berkstan　(b) webbase

図 5　各キャッシュ階層におけるキャッシュミスの回数



(a) berkstan　(b) webbase

図 6　1, 12, 24 スレッド実行時における消費メモリ帯域

ていない．RCM，LLP，Prop では圧縮を使用していることに注意されたい．いずれのグラフにおいてもスレッド数に対し提案手法の高速化率が最も高い．特に berkstan において Prop は線形に近いスケーラビリティを示している．また，berkstan と webbase の両方において Prop-NC より Prop が高いスケーラビリティを示しているのは圧縮による帯域消費削減の効果だと考えられる．ところが，圧縮を伴う提案手法も webbase においては 14 スレッド以降ほぼ性能向上がない．このことについては次のメモリ帯域消費の測定で議論する．webbase において LLP がスレッド数 12 とスレッド数 14 の間で大きく性能を劣化させているのは，スレッド 14 が NUMA ノード 2 に割り当てられることで生じるノード間の通信の影響だと考えられる．

6.2　キャッシュミス回数とメモリ帯域消費

まず 1 スレッド実行時にキャッシュミスを発生させた命令の数

表 3 評価結果

	(a) 24 並列時の PPR 処理時間 [秒]				(b) 圧縮によるサイズ削減率 (括弧内は近距離エッジ率)			(c) リオーダリング時間 [秒]			(d) n_Q
グラフ名	Naive	RCM	LLP	Proposed	RCM	LLP	Proposed	RCM	LLP	Proposed	Proposed
berkstan	0.36	0.13	0.15	0.07	36.2% (64.8%)	53.3% (89.6%)	58.2% (96.7%)	0.4	13.4	1.1	4
uk-2002	67.73	7.73	16.73	3.88	33.7% (56.0%)	48.8% (77.4%)	62.8% (97.2%)	18.1	1942.9	71.8	2
uk-2005	235.55	29.45	31.94	18.24	38.0% (58.4%)	57.3% (85.3%)	59.5% (88.3%)	52.2	6360.9	240.0	2
webbase	272.84	46.36	58.69	21.47	21.0% (46.4%)	50.5% (90.0%)	54.5% (96.0%)	109.3	9248.8	735.7	3
ljournal	9.21	3.61	4.22	2.94	5.4% (17.0%)	27.2% (48.2%)	32.5% (55.8%)	4.4	220.2	18.5	3
orkut	19.70	9.29	8.76	8.01	6.8% (11.0%)	20.1% (28.9%)	29.0% (40.9%)	7.1	729.9	26.9	3
patents	0.44	0.26	0.41	0.19	-10.4% (13.4%)	8.1% (43.8%)	19.4% (62.3%)	2.0	170.1	12.7	51

を計測した．結果を図 5 に示す．図の縦軸は対数である．階層的なクラスタを捉える提案手法はいずれのキャッシュ階層において最もキャッシュミス回数が少ない．特に L2 におけるキャッシュミスの少なさはキャッシュとクラスタそれぞれの階層を捉えたリオーダリングの効果だと考えられる．このことがメモリアクセスのレイテンシによるストールを減少させ高速な処理を可能にしている．また，グラフデータの圧縮によってより多くの情報をキャッシュに配置することが可能になるため，Prop は Prop-NC よりもキャッシュミス回数が少ない．さらに RCM と LLP を比較すると，グラフのクラスタ構造を捉える LLP は RCM よりもキャッシュミス回数が少ない．それにも関わらず図 3 において LLP の高速化率は RCM よりも低い．現状ではなぜこのような現象が発生するかを説明するための分析が不足しているため，今後更なる調査を行っていく．

次に berkstan と webbase について，1，12，24 スレッド時における消費メモリ帯域を計測した．結果を図 6 に示す．まず berkstan を見ると，24 スレッドにおける Prop の消費帯域の小ささが目立つ．12 スレッド時と比較して 24 スレッド時に Prop の帯域消費が小さいのは 2 つの NUMA ノードを使用しより大きなキャッシュ領域を使用可能になるためだと考えられる．一方で Prop-NC の帯域消費は高く，圧縮による帯域消費削減の効果を確認できる．次に webbase に目を向けると，berkstan よりもグラフが大規模であるため全体的にメモリ帯域消費が大きい．本論文の実験環境においてメモリ帯域がボトルネックとなる目安は，理論上の最大帯域の 75%である 44.8GB/s とされている [4]．つまり 12 スレッド時の Naive と Prop-NC，及び 24 スレッド時の Naive, Prop, Prop-NC はほぼ上限まで帯域を消費していることになり，図 4 (b) における性能向上の頭打ちと符合する．それに対し RCM と LLP はメモリ帯域を使い切らないまま性能向上が頭打ちとなっている．これはメモリアクセスのレイテンシの影響が支配的で，load 命令の発行までに時間がかかっていることが原因だと考えられる．

6.3 グラフの圧縮率

5.1 節で述べたグラフ圧縮手法の効果を表 3 (b) に示した．括弧内は近距離 CSR に格納されたエッジの割合である．なお 5.1 節にある通り CSR で用いられる 3 つの配列のうち AA は不要になるため，配列 JA と IA の和をグラフのサイズとした．提案手法と圧縮を組み合わせた場合最大で 62.8%のデータサイズを削減できている．また 3 つのリオーダリング手法を比較すると全てのグラフで提案手法が最も高い削減率を達成していることが分かる．つまり提案手法が最も効果的に近傍の頂点へ近い ID を与えることができているということを意味する．

6.4 リオーダリングに要する時間

各グラフのリオーダリング処理に要する時間を表 3 (c) に示す．提案手法は uk-2005 で約 4 分，1 億頂点規模の webbase では約 12 分でリオーダリングを完了している．LLP と比較すると uk-2002 などにおいて提案手法は約 27 倍高速である．

リオーダリングによる高速化では，リオーダリングに要した時間を後続する処理の高速化によって償却できるかどうかが重要である．そこでリオーダリング時間 (表 3 (c)) と 1 回の PPR 処理時間 (表 3 (a)) を比較する．提案手法によるリオーダリングに要する時間を t_R，Prop のクエリ処理時間を t_{RQ}，Naive のクエリ処理時間を t_{NQ}，リオーダリングに要する時間を償却するクエリ数を n_Q と置くとき，リオーダリングによって処理時間が短縮されることは次式によって表すことができる:

$$n_Q t_{NQ} > t_R + n_Q t_{RQ}$$

この不等式を満たす n_Q を表 3 (d) に示した．n_Q は最も小さい場合で uk-2005 と uk-2005 において 2，最も大きい場合においても patents の 51 である．検索や推薦など PPR の応用の多くは同じグラフに対して繰り返し問い合わせを行うため，リオーダリングに要した時間は容易に償却可能だと考えられる．

7. 結　　論

グラフ規模の増大に対応するため，効率的な並列処理によってマルチコア CPU の計算力を引き出し，高速なグラフ分析を可能にすることが求められている．グラフ処理では局所性の低いメモリアクセスが並列処理性能を劣化させることから，本論文では (i) 局所性を高めるリオーダリングと (ii) メモリ帯域消費を低減させるグラフデータ圧縮を前処理として実行することによってスケーラビリティを向上させる手法を提案する．提案手法は PPR のクエリ処理を RCM 比で最大 2 倍，LLP 比で最大 4 倍高速化することが実験により確認された．またリオーダリング処理は 1 億頂点規模のグラフでも約 12 分で完了することが可能であり，LLP と比べはるかに高速である．

今後は (i) リオーダリング後のグラフ処理に対するさらに詳細な性能評価，(ii) PPR 以外のグラフ処理に対してリオーダリングがもたらす効果の調査，および (iii) グラフ及びグラフ分析手法の性質がどのように提案手法の効果に影響しているかについての分析を行っていく．

文　献

[1] Intel® VTune™Amplifier XE. https://software.intel.com/en-us/intel-vtune-amplifier-x.

[2] Laboratory for Web Algorithmics. http://law.di.unimi.it/datasets.php.

[3] Stanford Large Network Dataset Collection. http://snap.stanford.edu/data/index.html.

[4] Using Intel® VTune™Amplifier XE to Tune Software on the Intel® Xeon® Processor E5/E7 v2 Family. https://software.intel.com/en-us/articles/using-intel-vtune-amplifier-xe-to-tune-software-on-the-intel-xeon-processor-e5e7-v2-family.

[5] E. Agirre and A. Soroa. Personalizing PageRank for Word Sense Disambiguation. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '09, pp. 33–41, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[6] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. In *Proceedings of the 20th international conference on World wide web - WWW '11*, p. 587, New York, New York, USA, Mar. 2011. ACM Press.

[7] U. Brandes. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[8] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A Portable Programming Interface for Performance Evaluation on Modern Processors. *Int. J. High Perform. Comput. Appl.*, 14(3):189–204, 2000.

[9] A. Buluç, S. Williams, L. Oliker, and J. Demmel. Reduced-Bandwidth Multithreaded Algorithms for Sparse Matrix-Vector Multiplication. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pp. 721–733, 2011.

[10] E. Cuthill and J. McKee. Reducing the Bandwidth of Sparse Symmetric Matrices. In *Proceedings of the 1969 24th National Conference*, ACM '69, pp. 157–172, New York, NY, USA, 1969. ACM.

[11] P. Desikan, N. Pathak, J. Srivastava, and V. Kumar. Incremental page rank computation on evolving graphs. In *Special interest tracks and posters of the 14th international conference on World Wide Web - WWW '05*, p. 1094, New York, New York, USA, May 2005. ACM Press.

[12] M. Frasca, K. Madduri, and P. Raghavan. NUMA-aware graph mining techniques for performance and energy efficiency. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pp. 95:1—-95:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[13] Y. Fujiwara, M. Nakatsuji, T. Yamamuro, H. Shiokawa, and M. Onizuka. Efficient personalized pagerank with accuracy assurance. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12*, p. 15, New York, New York, USA, Aug. 2012. ACM Press.

[14] J. A. George. *Computer Implementation of the Finite Element Method*. PhD thesis, Stanford, CA, USA, 1971.

[15] D. F. Gleich, L. Zhukov, and P. Berkhin. Fast Parallel PageRank: A Linear System Approach. Technical report, Yahoo! Research Labs, 2004.

[16] W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. Smith. Performance Modeling and Tuning of an Unstructured Mesh CFD Application. In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, SC '00, Washington, DC, USA, 2000. IEEE Computer Society.

[17] G. Jeh and J. Widom. Scaling personalized web search. In *Proceedings of the twelfth international conference on World Wide Web - WWW '03*, p. 271, New York, New York, USA, May 2003. ACM Press.

[18] A. D. K. Kaushik, B. D. E. Keyes, and B. F. S. D. Toward realistic performance bounds for implicit CFD codes. Proceedings of Parallel CFD '99, pp. 233–240. Elsevier, 1999.

[19] A. Kyrola, G. Blelloch, and C. Guestrin. GraphChi: Large-scale Graph Computation on Just a PC. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pp. 31–46, Berkeley, CA, USA, 2012. USENIX Association.

[20] Q. Liu, E. Chen, H. Xiong, and C. H. Ding. Exploiting user interests for collaborative filtering. In *Proceedings of the 19th ACM international conference on Information and knowledge management - CIKM '10*, p. 1697, New York, New York, USA, Oct. 2010. ACM Press.

[21] Y. Liu, X. Wang, J. Zhang, and H. Xu. Personalized PageRank Based Multi-document Summarization. In *Proceedings of the IEEE International Workshop on Semantic Computing and Systems*, WSCS '08, pp. 169–173, Washington, DC, USA, 2008. IEEE Computer Society.

[22] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. GraphLab: A New Framework for Parallel Machine Learning. *The 26th Conference on Uncertainty in Artificial Intelligence (UAI 2010)*, 2010.

[23] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. Berry. Challenges in Parallel Graph Processing. *Parallel Processing Letters*, 17(01):5–20, 2007.

[24] T. Maehara, T. Akiba, Y. Iwata, and K.-i. Kawarabayashi. Computing personalized pagerank quickly by exploiting graph structures. *Proceedings of the VLDB Endowment*, 7(12), 2014.

[25] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113, Feb. 2004.

[26] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. 1999.

[27] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic Multimedia Cross-modal Correlation Discovery. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pp. 653–658, New York, NY, USA, 2004. ACM.

[28] J. C. Pichel, D. E. Singh, and J. Carretero. Reordering Algorithms for Increasing Locality on Multicore Processors. In *2008 10th IEEE International Conference on High Performance Computing and Communications*, pp. 123–130. IEEE, Sept. 2008.

[29] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, May 2003.

[30] H. Tong, C. Faloutsos, and J.-Y. Pan. Fast Random Walk with Restart and Its Applications. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, pp. 613–622, Washington, DC, USA, 2006. IEEE Computer Society.

[31] 塩川, 藤原, 鬼塚. ノードの逐次集約による大規模グラフクラスタリングの高速化. In *Proceedings of the 4th Forum on Data Engineering and Information Managemen*, 2012.